# Comprehensive SBERT Fine-tuning Process for Bengali Text Classification

#### Overview

Based on the latest SBERT v5+ documentation, this process covers fine-tuning a SentenceTransformer model for your 10k Bengali customer support classification dataset using the modern (SentenceTransformerTrainer) approach.

#### **Step 1: Model Selection and Initialization**

#### Option A: Start with Bengali-Specific Model (Recommended)

python

from sentence\_transformers import SentenceTransformer

# Best for Bengali: Start with BanglaBERT base

model = SentenceTransformer("csebuetnlp/banglabert")

#### Option B: Start with Multilingual Model

python

# Alternative: Multilingual model with Bengali support

model = SentenceTransformer("paraphrase-multilingual-mpnet-base-v2")

#### Option C: Build from Transformer Base

python

from sentence\_transformers import SentenceTransformer, models

# Create from scratch using Bengali transformer

word\_embedding\_model = models.Transformer("csebuetnlp/banglabert", max\_seq\_length=256)

pooling model = models.Pooling(word embedding model.get word embedding dimension())

model = SentenceTransformer(modules=[word embedding model, pooling model])

### **Step 2: Dataset Preparation**

#### Format 1: Using SoftmaxLoss (Classification-Specific)

```
python
from datasets import Dataset
import pandas as pd
# Convert your data to the required format
data = {
  "sentence1": [], # First sentence from pair (can be empty string)
  "sentence2": [], # Your actual Bengali text
  "label": [] # Integer labels (0, 1, 2, ..., n_classes-1)
# Example with your Bengali data
train_data = []
for question, tag in your_data:
 # Create label mapping
 label_map = {
    'goodbye': 0,
   'greetings': 1,
    'agent_calling': 2,
    'repeat_again': 3,
    'namjari_process': 4,
   # ... etc for all your tags
 train_data.append({
    'sentence1': "", # Empty string for single sentence classification
    'sentence2': question, # Your Bengali text
    'label': label_map[tag]
 })
train_dataset = Dataset.from_list(train_data)
```

#### Format 2: Using MultipleNegativesRankingLoss (Contrastive Approach)

python			

## **Step 3: Loss Function Selection**

#### Option A: SoftmaxLoss (Direct Classification)

```
python

from sentence_transformers.losses import SoftmaxLoss

num_labels = len(set(your_label_values)) # Number of unique classes
embedding_dim = model.get_sentence_embedding_dimension()

loss = SoftmaxLoss(
model=model,
sentence_embedding_dimension=embedding_dim,
num_labels=num_labels,
concatenation_sent_rep=True,
concatenation_sent_difference=True,
concatenation_sent_multiplication=False
)
```

## Option B: MultipleNegativesRankingLoss (Contrastive Learning)

from sentence_transformers.losses import MultipleNegativesRankingLoss	
loss = MultipleNegativesRankingLoss(model=model, scale=20.0)	
Option C: CoSENTLoss (Modern Approach with Similarity Scores)	
you can generate similarity scores between pairs:	
python	
from sentence_transformers.losses import CoSENTLoss	
loss = CoSENTLoss(model=model, scale=20.0)	
Step 4: Training Arguments Configuration	
python	

```
from sentence_transformers import SentenceTransformerTrainingArguments
from sentence_transformers.training_args import BatchSamplers
args = SentenceTransformerTrainingArguments(
 # Required
 output_dir="./results/bengali-sbert-finetuned",
 # Core training parameters
 num_train_epochs=3, # Start with 3, adjust based on validation
 per_device_train_batch_size=16, # Adjust based on GPU memory
 per_device_eval_batch_size=16,
 learning_rate=2e-5, # Standard for transformer fine-tuning
 warmup_ratio=0.1,
 # Performance optimization
 fp16=True, # Use if your GPU supports it
 bf16=False, # Use if you have newer GPU (A100, etc.)
 gradient_checkpointing=True, # Save memory
 # For SoftmaxLoss, ensure each batch has examples from each class
 batch_sampler=BatchSamplers.GROUP_BY_LABEL if using_softmax else BatchSamplers.NO_DUPLICATES,
 # Evaluation and saving
 eval_strategy="steps",
 eval_steps=100,
 save_strategy="steps",
 save_steps=100,
 save_total_limit=2,
 load_best_model_at_end=True,
 metric_for_best_model="eval_loss",
 # Logging
 logging_steps=50,
 report_to=["tensorboard"], # or ["wandb"] if you use Weights & Biases
 run_name="bengali-sbert-classification",
 # Optional: For prompts (if using instruction-based training)
 # prompts="Classify this Bengali customer service query: ",
```

### **Step 5: Evaluation Setup**

```
python

from sentence_transformers.evaluation import EmbeddingSimilarityEvaluator, LabelAccuracyEvaluator

# For classification tasks

evaluator = LabelAccuracyEvaluator(
    sentences=eval_sentences,
    labels=eval_labels,
    name="bengali-classification-eval"

)
```

#### **Step 6: Training Execution**

```
python

from sentence_transformers import SentenceTransformerTrainer

trainer = SentenceTransformerTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset, # Optional
    loss=loss,
    evaluator=evaluator, # Optional
)

# Start training
trainer.train()

# Save the final model
model.save_pretrained("./bengali-sbert-final")
```

## **Step 7: Advanced Configurations**

#### A. Data Augmentation (Recommended for 10k samples)

python	

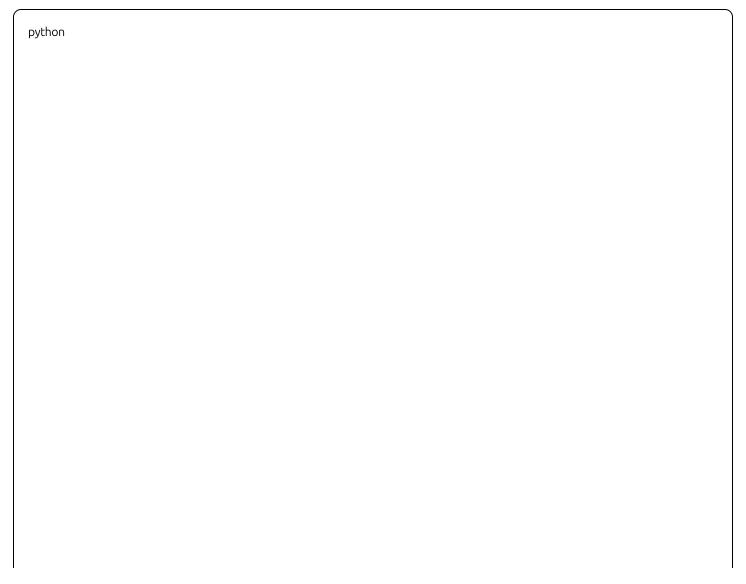
```
# Create synthetic examples through back-translation or paraphrasing
from transformers import pipeline

# Example: Use mT5 for Bengali paraphrasing
paraphraser = pipeline("text2text-generation", model="google/mt5-base")

def augment_data(text):
    prompt = f"paraphrase: {text}"
    return paraphraser(prompt, max_length=100)[0]['generated_text']

# Apply to your dataset
augmented_data = []
for item in original_data:
    augmented_data.append(item) # Original
    augmented_text = augment_data(item['sentence2'])
    augmented_data.append({**item, 'sentence2': augmented_text})
```

## B. Hyperparameter Optimization



```
from sentence_transformers.trainer import SentenceTransformerTrainer
def model_init():
 return SentenceTransformer("csebuetnlp/banglabert")
def hp_space(trial):
 return {
   "learning_rate": trial.suggest_float("learning_rate", 1e-5, 5e-5, log=True),
   "per_device_train_batch_size": trial.suggest_categorical("per_device_train_batch_size", [8, 16, 32]),
   "num_train_epochs": trial.suggest_int("num_train_epochs", 2, 5),
trainer = SentenceTransformerTrainer(
 model_init=model_init,
 args=args,
 train_dataset=train_dataset,
 eval_dataset=eval_dataset,
 loss=loss,
# Run hyperparameter optimization
best_trial = trainer.hyperparameter_search(
 hp_space=hp_space,
 compute_objective=lambda metrics: metrics["eval_loss"],
 n_trials=20,
 direction="minimize",
```

## C. Multi-dataset Training (If you have additional datasets)

python

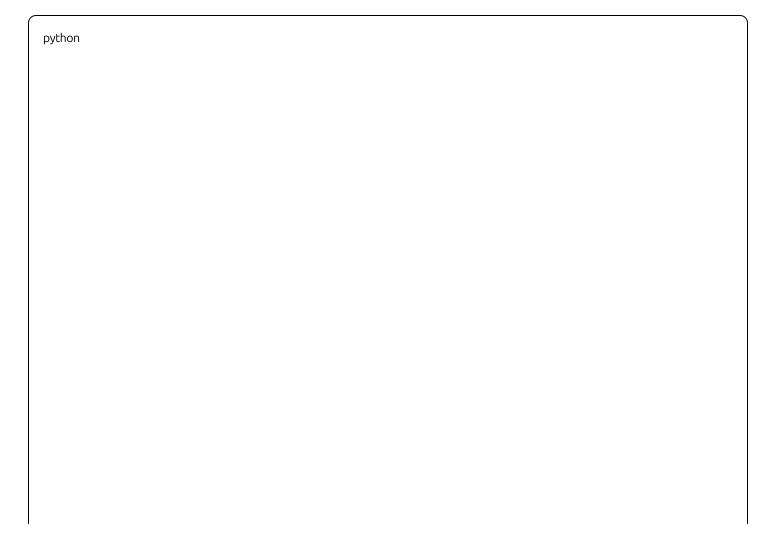
```
# Combine multiple datasets

train_datasets = {
    "main": main_train_dataset,
    "auxiliary": auxiliary_dataset, # Could be translated NLI data
}

# Different loss functions for different datasets
losses = {
    "main": SoftmaxLoss(model, embedding_dim, num_labels),
    "auxiliary": MultipleNegativesRankingLoss(model),
}

trainer = SentenceTransformerTrainer(
    model=model,
    train_dataset=train_datasets,
    loss=losses,
    args=args,
)
```

# Step 8: Model Usage After Training



```
# Load your fine-tuned model
model = SentenceTransformer("./bengali-sbert-final")
# For classification, you'll need to add a classifier on top
from sklearn.linear model import LogisticRegression
import numpy as np
# Generate embeddings for your training data
train_embeddings = model.encode(train_texts)
train_labels = your_train_labels
# Train classifier
classifier = LogisticRegression()
classifier.fit(train embeddings, train labels)
# For prediction
def classify_text(text):
  embedding = model.encode([text])
  prediction = classifier.predict(embedding)[0]
  confidence = classifier.predict_proba(embedding)[0].max()
  return prediction, confidence
# Example usage
text = "আমি নামজারি করতে চাই"
label, confidence = classify text(text)
print(f"Predicted: {label}, Confidence: {confidence:.3f}")
```

#### Key Considerations for Bengali

- 1. **Tokenization**: BanglaBERT handles Bengali tokenization properly
- 2. **Sequence Length**: Monitor your text lengths, adjust (max\_seq\_length) if needed
- 3. Mixed Script: Handle mixed Bengali-English text if present in your data
- 4. Class Imbalance: Use appropriate sampling strategies if classes are imbalanced

#### **Performance Tips**

- 1. Batch Size: Start with 16, increase if GPU memory allows
- 2. Learning Rate: 2e-5 is generally good, but try 1e-5 for stability
- 3. **Epochs**: 3-5 epochs usually sufficient for fine-tuning
- 4. Validation: Always use a held-out validation set

5. **Early Stopping**: Monitor validation loss to prevent overfitting

# **Expected Results**

With 10k examples and proper fine-tuning:

• Classification accuracy: 85-92% (depending on task complexity)

• Training time: 2-4 hours on single GPU

• Model size: ~440MB for BanglaBERT base